Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

Кафедра математической
кибернетики и компьютерных наук

АЛГОРИТМЫ ПРЕОБРАЗОВАНИЯ РЕГУЛЯРНЫХ ВЫРАЖЕНИЙ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

Студентки 4 курса 411 группы
направления 02.03.02 — Фундаментальная информатика и информационные технологии
факультета КНиИТ
Лукъяновой Анастасии Валерьевны

Научный руководитель
зав. кафедрой, к. ф.-м. н. _____ С. В. Миронов

Заведующий кафедрой
к. ф.-м. н. _____ С. В. Миронов

Саратов 2016

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 3
1 Регулярные языки и автоматы 5
1.1 Построение автомата по регулярному выражению 6
1.2 Построение регулярного выражения по автомату 6
1.3 Преобразования конечных автоматов 7
1.3.1 Детерминирование автомата 7
1.3.2 Минимизация автомата 7
2 Реализация преобразований автоматов и регулярных выражений на языке LISP 9
2.1 Представление данных 9
2.2 Построение автомата по регулярному выражению 10
2.3 Построение регулярного выражения по автомату 11
2.3.1 Минимизация автомата 11
2.4 Преобразование регулярных выражений 12
ЗАКЛЮЧЕНИЕ 13
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 14

ВВЕДЕНИЕ

В настоящее время аппарат регулярных выражений внедряется повсеместно. Наиболее часто решаемые задачи с помощью регулярных выражений — задачи описания лексики формального языка и задачи поиска и фильтрации в тексте [1, 2].

Регулярные выражения для задач фильтрации могут иметь значительную сложность с точки зрения таких характеристик как длина выражения и высота (звездная высота) выражения [3, 4]. Если с истечением времени в некоторый фильтр, заданный регулярным выражением, добавляются дополнительные условия, то окончательное регулярное выражение может принимать нечитабельную форму.

Существует несколько теоретических подходов к сокращению значений величин высоты или длины регулярных выражений. Но ни один подход на сегодняшний день не дает гарантированного результата сокращения [5].

В настоящей работе ставится задача подготовить базу для экспериментов по сокращению регулярных выражений на языке LISP, а именно, реализовать методы получения конечных автоматов по регулярным выражениям и получения регулярного выражения по автомату. В качестве входных регулярных выражений должны выступать строки, содержащие регулярные выражения в записи, принятой в [6].

Для выполнения данной задачи потребуется:

- определить структуры данных для представления регулярных выражений и конечных автоматов на языке LISP;
- реализовать алгоритмы построения конечных автоматов для результатов регулярных операций (конкатенации, итерации и объединения) над автоматными языками;
- реализовать алгоритм построения детерминированного автомата по недетерминированному;
- реализовать алгоритм построения минимального детерминированного автомата по заданному автомату;
- реализовать алгоритм исключения состояний автомата для построения регулярного выражения по заданному автомату.

Данная работа состоит из введения, двух глав, заключения, списка литературы и одного приложения. Введение содержит краткое описание и основ-

ную задачу, которую требуется выполнить.

В первой главе объясняются основные определения и понятия, которые используются во всей работе, и рассматриваются теоретические аспекты следующих алгоритмов:

— алгоритм построения автомата по регулярному выражению;
— алгоритм построения регулярного выражения по автомату;
— алгоритмы преобразования конечных автоматов.

Во второй главе подробно рассматривается представление данных в программной реализации, описана работа алгоритмов с регулярными выражениями и автоматами, а так же приведены примеры работы реализованных алгоритмов.

1 Регулярные языки и автоматы

Регулярное множество имеет следующее определение:

— $\oslash$ — регулярное множество

— $\{\varepsilon\}$ — регулярное множество

— Если a является символом некоторого алфавита, то $\{a\}$ – регулярное множество.

— Если L_1 и L_2 — регулярные множества, то $L_1 \cap L_2$, $L_1 \cup L_2$, $L_1 L_2$, L^* и $\bar{L}$ — регулярные множества.

Регулярные выражения — это структурные записи для описания регулярных множеств [6]. Регулярные выражения имеют следующее обозначения:

— $\varnothing$ — регулярное выражение для пустого множества.

— ε — регулярное выражение для множества $\{\varepsilon\}$.

— Если символ a — символ некоторого алфавита, то a — регулярное выражение для регулярного множества $\{a\}$.

— Если α — регулярное выражение для множества A, а β — регулярное выражение для регулярного множества B, то $\alpha + \beta$, $(\alpha)(\beta)$, $(\alpha)^*$ — регулярные выражения для регулярных множеств $A \bigcup B$, AB, A^* соответственно.

Конечным автоматом называют пятёрку $M = (Q, \Sigma, \delta, S, F)$, где

— Q — множество имён состояний;

— Σ — конечный входной алфавит;

— $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ — функция перехода;

— $S \subseteq Q$ — множество начальных состояний автомата;

— $F \subseteq Q$ — множество заключительных состояний.

Конечный детерминированный автомат отличается от недетерминированного тем, что из каждого состояния автомат может перейти по определённому входному сигналу точно в одно другое состояние. Кроме того, в детерминированном автомате множество начальных состояний содержит в точности одно состояние.

Пусть $M = (Q, \Sigma, \delta, S, F)$ — конечный недетерминированный автомат. Для построения конечного детерминированного автомата
$M' = (Q', \Sigma, \delta', \{q'_0\}, F')$ применяются следующие правила:

— $Q' = \mathcal{P}(Q)$;

— $\delta'(q', a) = \{r'\}$, где $r' = \bigcup\limits_{q \in q'} \delta(q, a), \forall q' \in Q', \forall a \in \Sigma$;

– $q'_0 = S$;
– $F' = \{q' \in Q' \mid (q' \cap F) \neq \varnothing\}$.

Конечные автоматы содержат в себе набор состояний и переходов между ними, которые зависят от поданных на вход данных.

Согласно [6] класс автоматных языков (языков, допускаемых конечными автоматами) совпадает с классом регулярных множеств. Таким образом, для любого конечного автомата можно найти регулярное выражение, определяющее тот же язык, и для любого регулярного выражения можно найти соответствующий конечный автомат.

1.1 Построение автомата по регулярному выражению

Для построение автомата по регулярному выражению рассмотрим два конечных автомата M_1 и M_2. Пусть $L_1 = L(M_1)$ и $L_2 = L(M_2)$.

1. построение автомата для объединения языков ($L_1 \cup L_2$);
2. построение автомата для конкатенации языков ($L_2 L_1$);
3. построение автомата для итерации языка (L_1^*).

1.2 Построение регулярного выражения по автомату

Метод исключения состояний позволяет построить регулярное выражение α по полученному на входе конечному автомату $M = (Q, \Sigma, \delta, S, F)$, такое, что $L(\alpha) = L(M)$.

Дан автомат $M = (Q, \Sigma, \delta, S, F)$, представленный в виде диаграммы переходов. В этой диаграмме каждую дугу помечаем регулярным выражением, состоящим из объединения входных сигналов, приписанных дуге.

Исключение некоторого состояния q_2 заключается в следующем. В автомате рассматриваются все тройки состояний, соединенные дугами. Тогда при исключении состояния q_2 нужно изменить метку у дуги между q_1 и q_3 на выражение $\alpha + \beta\gamma^*\sigma$. Состояние q_2 исключается, когда будут обработаны таким образом все подобные пути, проходящие через q_2.

Заметим, что какие-то два состояния в тройке могут совпадать, а некоторые из дуг отсутствовать. Если некоторые (или все) из дуг отсутствуют, то можно считать, что они присутствуют, с приписанным к ним регулярным выражением $\varnothing$.

Выдать регулярное выражение $\sum\limits_{s \in S, q \in F} \alpha_{sq}$ и закончить алгоритм (здесь под знаком $\sum$ предполагается объединение регулярных выражений).

1.3 Преобразования конечных автоматов

1.3.1 Детерминирование автомата

В теории конечных автоматов сказано [1], что для каждого недетерминированного конечного автомата существует детерминированный конечный автомат, который принимает тот же язык регулярных выражений, что и недетерминированный конечный автомат. Для преобразования НКА в ДКА, необходимо сначала обговорить, что алфавиты этих двух автоматов одинаковы, а так же начальное состояние детерминированного автомата является множеством содержащим все начальные состояния недетерминированного автомата [6].

Для реализации алгоритма детерминирования автомата, нам необходим конечный недетерминированный автомат вида $M = (Q, \Sigma, \delta, S, F)$. Результатом выполнения данного алгоритма будет автомат вида $M' = (Q', \Sigma, \delta', \{q_0\}, F')$.

- Сначала строится множество состояний $Q' = \mathcal{P}(Q)$, где $\mathcal{P}(Q)$ — множество всех подмножеств множества Q.
- Функция переходов определяется следующим образом: $\delta'(q', a) = r$, где $r' = \bigcup\limits_{q \in q'} \delta(q, a)$, $\forall q' \in Q'$, $\forall a \in \Sigma$.
- Назначается новое начальное состояние ${q_0}' = S$.
- Строится множество заключительных состояний $F' = \{q' \in Q' \mid (q' \cap F) \neq \varnothing$.

1.3.2 Минимизация автомата

Минимизация детерминированного конечного автомата подразумевает нахождение для каждого ДКА эквивалентного детерминированного конечного автомата с наименьшим числом состояний. При этом основная идея минимизации строится на идее эквивалентности состояний — то есть на возможности заменить два эквивалентных состояния одним. Поэтому при реализации этого алгоритма на вход подаётся конечный детерминированный автомат вида $M = (Q, \Sigma, \delta, \{q\}, F)$, а результатом будет автомат $M' = (Q', \Sigma, \delta', \{{q_0}'\}, F')$ такой, что $L(M) = L(M')$ и Q' содержит минимальное возможное количество состояний.

Алгоритм выглядит следующим образом:

1. Из данного автомата удаляются все недостижимые состояния. Получается $M_1(Q_1, \Sigma, \delta_1, \{q_1\}, F_1)$.
2. Множество состояний Q_1 разбивается на классы эквивалентности, со-

ответствующие отношению неразличимости $\equiv$ (с использованием алгоритма заполнения таблицы) и $[Q_1]_{\equiv}$ — совокупность полученных классов эквивалентности.

3. Следующим образом определяется автомат $M' = (Q', \Sigma, \delta', \{q_0{}'\}, F')$, где
   - $Q' = [Q_1]_{\equiv}$
   - $q_0{}' = [q_0]_{\equiv}$
   - $F' = \{[q]_{\equiv} \in [Q_1]_{\equiv} | q \in Q_1 \cap F\}$
   - $\delta'([q]_{\equiv}, a) = \{[r]_{\equiv} \in [Q_1]_{\equiv} | r \in \delta(q, a)\}, q \in Q_1, a \in \Sigma$.

   M' выдается в качестве результата и алгоритм завершается.

Разбиение множества состояний автомата на классы эквивалентности будем осуществлять с помощью метода заполнения таблицы.

Для алгоритма заполнения таблицы требуется конечный автомат вида $M = (Q, \Sigma, \delta, \{q\}, F)$, в котором $n = \#Q$ — количество состояний данного автомата. Результатом преобразований данного алгоритма будет $T - n \times n$ — таблица, ячейки которой обозначаются следующим образом: $T(q, r)$, где q и r — обозначают состояния. Если состояния q и r различимы, то в ячейку, которая стоит на пересечении этих двух состояний заносится X.

2 Реализация преобразований автоматов и регулярных выражений на языке LISP

Для программной реализации алгоритмов преобразования регулярных выражений использовался язык Lisp [7–9]. Разработка проводилась с использованием интегрированной оболочки LispWorks.

2.1 Представление данных

В качестве входных данных для программы будем использовать строки, в которых записаны элементы регулярных выражений по следующим правилам:

1. любой символ, отличный от «*», «+», «(», «)» и пробела является символом языка, определенного регулярным выражением;
2. пробелы при чтении регулярного выражения игнорируются;
3. символы круглых скобок «(» и «)» обозначают ограниченное регулярное выражение, записанное в скобках. Пустые скобки обозначают регулярное выражение, определяющее язык, состоящий из одного пустого слова;
4. несколько выражений, записанных подряд, задают конкатенацию определяемых ими языков;
5. два выражения, записанных через символ «+», задают объединение определяемых ими языков;
6. символ «*», указанный после символа или после выражения в скобках, задает итерацию языка, составленного из односимвольного слова (составленного из этого символа) или определенного регулярным выражением в скобках.
   Например

```
"(00+11)*((01+10)(11+00)*(01+10)(11+00)*)*"
```

представленное регулярное выражение определяет язык, состоящий из цепочек нулей и единиц, в которых четное число нулей и четное число единиц.

Конечный автомат будем представлять в виде списка из трех элементов в соответствующем порядке:

1. таблица переходов, представляющая из себя список, в котором каждый элемент-тройка представлен в виде списка
   `(состояние1 символ состояние2)`,
   указывающего, что в автомате присутствует переход из состояния с номером `состояние1` в состояние с номером `состояние2` по входному сигналу

символ;

2. список начальных состояний автомата;
3. список заключительных состояний автомата.

Например

```
(((1 0 4) (1 1 2) (2 0 3) (2 1 1) (3 0 2) (3 1 4) (4 0 1) (4 1 3))
 (1)
 (2))
```

— конечный детерминированный автомат, допускающий цепочки из нулей и единиц, в которых четное число нулей и нечетное число единиц.

2.2 Построение автомата по регулярному выражению

Автомат, определяющий тот же язык, что и регулярное выражение, будем строить из автоматов для элементарных языков, последовательно строя автоматы, для результатов регулярных операций, упомянутых в регулярном выражении.

Реализация алгоритма объединения (функция `ob`) выглядит следующим образом:

1. Получаем на вход два автомата.
2. Переименовываем все состояния второго автомата, увеличивая каждое состояние на максимальный элемент первого автомата.
3. Объединяем функции переходов автоматов.
4. Объединяем начальные состояния автоматов.
5. Объединяем конечные состояния автоматов.

Реализация алгоритма конкатенации (функция `conc`) выглядит следующим образом:

1. На вход подаётся два автомата.
2. Переименовываем все состояния второго автомата, увеличивая каждое состояние на максимальный элемент первого автомата.
3. Находим все переходы второго автомата из его начального состояния. Повторяем найденные переходы для каждого заключительного состояния первого автомата.
4. Создаём новую таблицу переходов, объединяя переходы первого и второго автомата с новыми переходами.
5. В качестве начальных состояний нового автомата берем все начальные состояния первого автомата;

6. Если пересечение множеств начальных и конечных состояний второго автомата не пусто, то заключительными состояниями объявляем объединение множеств заключительных состояний двух автоматов, иначе только заключительные состояния второго автомата.

При построении автомата для итерации автоматного языка (функция `iter`) выполняются следующие действия:

1. Подаётся на вход автомат.
2. Находятся все переходы из начального состояния автомата. Найденные переходы дублируются для каждого заключительного состояния автомата.
3. Создаётся новое состояние (увеличивается количество состояний автомата).
4. Новое состояние объявляется дополнительным начальным.
5. Новое состояние объявляется дополнительным заключительным.

2.3 Построение регулярного выражения по автомату

Для преобразования ДКА в регулярное выражение (функция `aut->reg`) мы используем метод исключения состояний. Он состоит в том, что мы исключаем некоторое состояние и все пути автомата, что проходят через это состояние. Однако, чтобы язык автомата остался без изменений, нам необходимо скорректировать метки на путях, обходящих исключаемое состояние. При этом вместо обычных меток, на дугах, описывающих такие пути, будем указывать регулярные выражения, описывающие множество слов, по которым осуществляется переход. Таким образом мы получаем в рассмотрение автомат, у которого метками переходов являются регулярные выражения [6].

2.3.1 Минимизация автомата

Минимизация детерминированного конечного автомата подразумевает нахождение для каждого ДКА эквивалентный детерминированный конечный автомат с наименьшим числом состояний. При этом основная идея минимизации строится на идее эквивалентности состояний, то есть на возможности заменить два различных состояния одним.

Алгоритм минимизации детерминированного конечного автомата (функция `minimize`) выглядит следующим образом:

— Для обнаружения всех эквивалентных состояний конечного автомата применяется алгоритм заполнения таблицы, который состоит в рекурсивном обнаружении пар различимых состояний.
— Множество состояний подвергается разбиению на классы эквивалентности, то есть на подмножества, в которых любая пара состояний (p, q) не различима.
— Строится новый ДКА с минимальным числом состояний, используя в качестве состояний полученные классы эквивалентности. Если S — это класс эквивалентности, $q \in S$ и $\delta(q, a) = r$ то δ' — функция переходов нового автомата, определяется как $\delta'(S, a) = T$, где T — класс эквивалентности, содержащий r.
— Так же необходимо отметить, что начальным состоянием является класс эквивалентности, который содержит в себе начальное состояния старого автомата.

2.4 Преобразование регулярных выражений

Для удобочитаемости регулярных выражений был реализован алгоритм их упрощений (функция `simplify-reg`).

Перечислим основные принципы, включенные в алгоритм. Пусть α, β И γ — регулярные выражения. Тогда

— $\alpha\alpha = \alpha$
— $\alpha + \alpha = \alpha$
— $\alpha + \varepsilon = \varepsilon + \alpha = \alpha$
— $\alpha\varepsilon = \varepsilon\alpha = \alpha$
— $(\alpha) = \alpha$
— $\alpha(\beta\gamma) = (\alpha\beta)\gamma = \alpha\beta\gamma$
— $(\alpha\beta)\gamma = \alpha\beta\gamma$
— $(\varepsilon)* = \varepsilon$

ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломной работы был реализован набор функций для работы с регулярными выражениями и конечными автоматами в терминах теории формальных языков. Данный пакет предназначен для дальнейшего изучения аппарата регулярных выражений на предмет минимизации параметров, таких как длина выражения и его звездная высота.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Теория синтаксического анализа, перевода и компиляции / Под ред. А. Ахо, Дж. Ульман. — Москва: Мир, 1978. — Т. 1.

2 Компиляторы: принципы, технологии, инструменты / Под ред. А. Ахо, Р. Сети, Дж. Ульман. — Москва — Санкт-Петербург — Киев: Вильямс, 2003.

3 *Баумгертнер, С. В.* Мультиэвристический подход к проблеме звездно-высотной минимизации недетерминированных конечных автоматов / С. В. Баумгертнер, Б. Ф. Мельников // *Вестник ВГУ. Серия: системный анализ и информационные технологии*. — 2010. — № 1. — С. 5–7.

4 *Хомякова, Е. С.* Организация подсчета обобщенной звездной высоты регулярного языка / Е. С. Хомякова, С. В. Миронов // *Компьютерные науки и информационные технологии. Материалы Междунар. науч. конф.* — 2014. — С. 358–360.

5 Жемчужины теории формальных языков / Под ред. А. Саломаа. — Москва: Мир, 1986.

6 Введение в теорию автоматов, языков и вычислений / Под ред. Дж. Хопкрофт, Р. Мотвани, Дж. Ульман. — Москва — Санкт-Петербург — Киев: Вильямс, 2002.

7 *Грэм, П.* ANSI Common Lisp / П. Грэм. — СПб.: Символ-Плюс.

8 Common Lisp HyperSpec. [Электронный ресурс]. — URL: `http://www.lispworks.com/documentation/HyperSpec/Front/index.htm` (Дата обращения 21.06.2016). Загл. с экр. Яз. англ.

9 Очень краткое введение в язык Лисп [Электронный ресурс]. — URL: `http://homelisp.ru/help/lisp.html#u10` (Дата обращения 21.06.2016). Загл. с экр. Яз. рус.